Algorithm: Lab10 (By Sujiv Shrestha ID:610145)
Problem 1.
1. Below, the BinarySearch and Recursive Fibonacci algorithms are shown. In each case, what are the subproblems? Why do we say that the subproblems of BinarySearch *do not overlap* and the subproblems of Recursive Fibonacci *overlap?* Explain.

```
Algorithm binSearch(A, x, lower, upper)
    Input: Already sorted array A of size n, value x to be
           searched for in array section A[lower]..A[upper]
    Output: true or false

if lower > upper then return false
mid ← (upper + lower)/2
if x = A[mid] then return true
if x < A[mid] then
      return binSearch(A, x, lower, mid – 1)
else
      return binSearch(A, x, mid + 1, upper)
```

```
Algorithm fib(n)
    Input: a natural number n
    Output: F(n)

if (n = 0 || n = 1) then return n

return fib(n-1) + fib(n-2)
```

Answer: The subproblems in binary search algorithm is searching given value x in array A in the range given by lower and upper indices. In binary search the value x is searched in array A by dividing it into two halves and deciding which half to search for x so the subproblems do not overlap.

While in recursive Fibonacci algorithm the subproblem is to calculate two previous terms of Fibonacci series as the term in Fibonacci series is the sum of its two previous terms. So, in calculating the Fibonacci series the subproblems overlap as the calculation of first previous term involve calculation of the second previous term itself.

Problem2
2. Consider the following instance of the Edit Distance problem: EditDistance(“maple”, “kale”). Taking the iterative dynamic programming approach to solve this problem, fill out the values in the table.

D	“”	“k”	“ka”	“kal”	“kale”
“”	0	1	2	3	4
“m”	1	1	2	3	4
“ma”	2	2	1	2	3
“map”	3	3	2	3	3
“mapl”	4	4	3	2	3
“maple”	5	5	4	3	3

Problem3

3. (Interview Question) Devise a dynamic programming solution for the following problem:
Given two strings, find the length of longest subsequence that they share in common.
Different between substring and subsequence:
Substring: the characters in a substring of S must occur contiguously in S.
Subsequence: the characters can be interspersed with gaps.
For example: Given two Strings - "regular" and "ruler", your algorithm should output 4.

Algorithm longestSub(S1, S2)
Input string S1 or length n, string S2 of length m
Output length of longest subsequence between S1 and S2
D[n][m]←two dimensional array of size nXm initialized with -1
return recurseLongSub(S1,S2, |S1|, |S2|, D)

Algorithm recurseLongSub(S1, S2, i, j, D)
Input string S1, string S2, integer i pointing end of prefix in S1, integer j pointing end of prefix in S2, two dimensional memo array D of size |S1|X|S2|
Output length of longest subsequence between $S1_i$ and $S2_j$

if(i=0 or j=0) then return 0

if(S1[i] = S2[j]) then
    if(D[i-1][j-1]=-1) then
        D[i-1][j-1] = recurseLongSub(S1, S2, i-1, j-1, D)
    D[i][j] = D[i-1][j-1]+1
else
    if (D[i-1][j]=-1) then
        D[i-1][j] = recurseLongSub(S1, S2, i-1, j, D)
    if(D[i][j-1]=-1) then
        D[i][j-1] = recurseLongSub(S1, S2, I, j-1, D)
    D[i][j] = max(D[i-1][j], D[i][j-1]
return D[i][j]

Java Implementation:

```
	public static int longestSub(String S1, String S2) {
		int[][] D = new int[S1.length()][S2.length()];
		for(int i=0;i<S1.length();i++) {
			for(int j=0;j<S2.length();j++) {
				D[i][j] = -1;
			}
		}
		int result = recurseLongSub(S1,S2,S1.length()-1,S2.length()-1,D);
		for(int i=0;i<S1.length();i++)
			System.out.println(Arrays.toString(D[i]));
		return result;
//		return D[S1.length()-1][S2.length()-1];
	}
```

```
private static int recurseLongSub(String s1, String s2, int i, int j, int[][] D)
{
	if(i==0||j==0) {
		return 0;
	}

	if(s1.charAt(i-1)==s2.charAt(j-1)) {
		if(D[i-1][j-1]==-1) {
			D[i-1][j-1]=recurseLongSub(s1,s2,i-1,j-1,D);
		}
		D[i][j]=D[i-1][j-1]+1;
	}
	else {
		if(D[i-1][j]==-1)
			D[i-1][j]=recurseLongSub(s1,s2,i-1,j,D);
		if(D[i][j-1]==-1)
			D[i][j-1]=recurseLongSub(s1,s2,i,j-1,D);
		D[i][j] = Math.max(D[i-1][j], D[i][j-1]);
	}
	return D[i][j];
}
```

Test:

```
System.out.println("Result:"+ longestSub("regular","rular"));
```

```
Output:
(Memo matrix)
[0, -1, -1, -1, -1]
[-1, 1, -1, -1, -1]
[0, 1, -1, -1, -1]
[0, 1, -1, -1, -1]
[-1, -1, 2, -1, -1]
[-1, -1, -1, 3, -1]
[-1, -1, -1, -1, 4]

Result:4
```

Problem4

4. *(Optional Interview Question)* Devise a dynamic programming solution for the following problem:

Given a positive integer n, find the least number of perfect square numbers which sum to n. (Perfect square numbers are 1, 4, 9, 16, 25, 36, 49, …)

For example, given n = 12, return 3; (12 = 4 + 4 + 4)

Given n = 13, return 2; (13 = 4 + 9)

Given n = 67 return 3; (67 = 49 + 9 + 9)

Algorithm leastSqSum(n)

Input integer n

Output least number of perfect square numbers whose sum is n

finalAns←new list of integer

min←maximum possible value of integer

for(i=1 to square root of n) **do**

temp←new list

temp.add(i)

sqTerm(n-i*i, temp)

```
            if(min>|temp|) then
                  finalAns←temp
                  min←|finalAns|

      return finalAns

Algorithm sqTerm(n, lst)
      Input integer n, list of integer lst
      Output list of integers whose perfect square numbers sum is n
      if(n=0) then
            return lst
      sq = square root of n
      lst.add(sq)
      if(sq*sq=n) then
            return lst
      else
            return sqTerm(n-sq*sq, lst)
```

Java implementation

```java
private static int leastSqSum(int n) {
      List<Integer> finalAns = new ArrayList<>();
      int min = Integer.MAX_VALUE;
      for(int i=1;i*i<=n;i++) {
            List<Integer> temp = new ArrayList<Integer>();
            temp.add(i);
            temp = sqTerm(n-i*i, temp);
            if(min>temp.size()) {
                  finalAns = temp;
                  min = finalAns.size();
            }
      }
      System.out.println(finalAns);
      return finalAns.size();
}

private static List<Integer> sqTerm(int n, List<Integer> lst) {
      //System.out.println(n);
      if(n==0)
            return lst;
      int sq = (int) Math.sqrt(n);
      lst.add(sq);
      if(sq*sq==n) {
            return lst;
      }
      else
            return sqTerm(n-sq*sq, lst);
}
```